Today: Covering Problem

- Implement a "gate-level" netlist in terms of some library of primitives
- General Formulation
  - Make it easy to change technology
  - Make it easy to experiment with library requirements
  - Evaluate benefits of new cells...
  - Evaluate architecture with different primitives

Elements of a library - 1

<table>
<thead>
<tr>
<th>Element/Area Cost</th>
<th>Tree Representation (normal form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVERTER</td>
<td>![INVERTER Diagram]</td>
</tr>
<tr>
<td>NAND2</td>
<td>![NAND2 Diagram]</td>
</tr>
<tr>
<td>NAND3</td>
<td>![NAND3 Diagram]</td>
</tr>
<tr>
<td>NAND4</td>
<td>![NAND4 Diagram]</td>
</tr>
</tbody>
</table>

Input Circuit Netlist

``subject DAG``

- Each wire is a network (net).
- Each net has a single source (the gate that drives it).
- In general, net may have multiple sinks (gates that take as input)
Input Circuit Netlist

```
subject DAG
```

- A list of the nets (netlist) fully describes the circuit
  0 nand 1 6
  1 inv 2
  2 nand 3 4

Problem Statement

Find an "optimal" (in area, delay, power) mapping of this circuit (DAG)

What's the problem?

Trivial Covering

Cost Models

Cost Model: Area

- **Assume**: Area in gates
- or, at least, can pick an area/gate
  - so proportional to gates
- *e.g.*
  - Standard Cell design
  - Standard Cell/route over cell
  - Gate array

Why covering now?

- Nice/simple cost model
- Problem can be solved well
  - somewhat clever solution
- General/powerful technique
- Show off special cases
  - harder/easier cases
- Show off things that make hard
- Show off bounding
Standard Cells
• Lay out gates so that heights match
  – Rows of adjacent cells
  – Standardized sizes
• Motivation: ease place and route

Standard Cell Area

Cost Model: Delay
• Delay in gates
  – at least assignable to gates
    • $T_{wire} \ll T_{gate}$
    • $T_{wire} \approx$ constant
  – delay exclusively/predominantly in gates
    • Gates have $C_{out}, C_{in}$
    • lump capacitance for output drive
    • delay $\approx T_{gate} + \text{fanout} \times C_{in}$
    • $C_{wire} \ll C_{in}$
    • or $C_{wire}$ can lump with $C_{out}/T_{gate}$

Logic Delay

Parasitic Capacitances
**Delay of Net**

- Delay in gates
  - at least assignable to gates
  - \( T_{\text{wire}} \ll T_{\text{gate}} \)
  - \( T_{\text{wire}} \approx \text{constant} \)
  - delay exclusively/predominantly in gates
  - Gates have \( C_{\text{out}}, C_{\text{in}} \)
  - lump capacitance for output drive
  - delay \( \approx T_{\text{g}} + \text{fanout} \times C_{\text{in}} \)
  - or \( C_{\text{wire}} \) can lump with \( C_{\text{out}}/T_{\text{gate}} \)

**Cost Model: Delay**

- \( T_{\text{wire}}(300\mu m) \approx 1\text{ps} \)
- \( W_{\text{g}} = 0.3\mu m \)

**Cost Models**

- Why do I show you models?
  - not clear there’s one “right” model
  - changes over time
  - you’re going to encounter many different kinds of problems
  - want you to see formulations so can critique and develop own
  - simple cost models make problems tractable
  - are surprisingly adequate
  - simple, at least, help bound solutions
  - may be wrong today…need to rethink

**Approaches**

- Greedy work?
  - Greedy = pick next locally “best” choice

**Greedy In → Out**

- \( F = 22\text{nm CMOS} \)
  - \( T_{\text{g}}(\text{inv drive 4 inv}) \approx 1\text{ps} \)
  - \( T_{\text{wire}}(300\mu m) \approx 1\text{ps} \)
  - \( W_{\text{g}} = 0.3\mu m \)
Greedy In→Out

Greedy Out→In

But…

Greedy Problem

• What happens in the future (elsewhere in circuit) will determine what should be done at this point in the circuit.

• Can’t just pick best thing for now and be done.

Brute force?

• Pick a node (output)
• Consider
  – all possible gates which may cover that node
  – branch on all inputs after cover
  – pick least cost node
Pick a Node

Brute force?

- Pick a node (output)
- Consider
  - all possible gates which may cover that node
  - recurse on all inputs after cover
  - pick least cost node
- Explore all possible covers
  - can find optimum

Analyze brute force?

- Time?
  \[ T_{\text{Brute}}(\text{node}) = \max_{i < j} \left( T_{\text{match}}(P_i) + \max_{i < j} (T_{\text{Brute}}(in_j)) \right) \]
- Say \( P \) patterns, constant time to match each
  - (if patterns long could be \( > O(1) \))
- \( P \)-way branch at each node...
  - ...exponential
  - \( O(P^{\text{depth}}) \)

Structure inherent in problem to exploit?

- There are only \( N \) unique nodes to cover!

Structure

- If subtree solutions do not depend on what happens outside of its subtree
  - separate tree
  - farther up tree
- Should only have to look at \( N \) nodes.
- Time(\( N \)) = \( N^3P^2T(\text{match}) \)
  - w/ \( P \) fixed/bounded \( \Rightarrow \) linear in \( N \)
  - w/ cleverness work isn’t \( P^2T(\text{match}) \) at every node
Idea Re-iterated

- Work from inputs
- Optimal solution to subproblem is contained in optimal, global solution
- Find optimal cover for each node
- Optimal cover:
  - examine all gates at this node
  - look at cost of gate and its inputs
  - pick least

Work front-to-back

Work Example (area)

library
test

Work Example (area)

library
test

Work Example (area)
Work Example (area)

5 + 2 + 3 = 10

3 + 5 = 8

3 + 2 = 5

5 + 4 + 5 = 14
Work Example (area)

\[ 8 + 2 + 3 = 13 \]

\[ 13 + 2 = 15 \]

\[ 3 + 2 + 4 = 9 \]
Work Example (area)

\[9 + 4 + 3 = 16\]

Work Example (area)

\[8 + 2 + 4 + 4 = 18\]

Work Example (area)

\[16 + 2 = 18\]

Work Example (area)

\[13 + 5 + 4 = 22\]
Work Example (area)

- Example 1:
  - Equation: $18 + 3 = 21$
  - Area: $17$

- Example 2:
  - Equation: $9 + 4 + 4 = 17$
  - Area: $17$

Optimal Cover

- Optimal solution:
  - Area: $31$

- Solution:
  - Area: $17$

Much better than $31$!
Note

- There are nodes we cover which will **not** appear in final solution.

“Unused” Nodes

[Diagram of a circuit showing “Unused” Nodes]

Dynamic Programming Solution

- Solution described is general instance of dynamic programming
- Require:
  - optimal solution to subproblems is optimal solution to whole problem
  - (all optimal solutions equally good)
  - divide-and-conquer gets same (finite/small) number of subproblems
- Same technique used for instruction selection in code generation for processors

Delay

- Similar
  - Delay(node) = Delay(gate) + Max(Delay(input))

DAG

- DAG = Directed Acyclic Graph
  - Distinguish from tree (tree ⊂ DAG)
  - Distinguish from cyclic Graph
  - DAG ⊂ Directed Graph (digraph)

Trees vs. DAGs

- Optimal for trees
  - why?
    - Delay
    - Area
Not optimal for DAGs

- Why?

\[ 1 + 1 + 1 = 3 \]

Not optimal for DAGs (area)

- Why?

\[ \text{Cost}(N) = \text{Cost}(\text{gate}) + \sum \text{Cost}(\text{input nodes}) \]

- think of sets
- cost is magnitude of set union
- **Problem:** minimum cost (magnitude) solution isn’t necessarily the best pick
  - get interaction between subproblems
  - subproblem optimum not global...

DAG Example

- Cover with 3 input gates

\[ 1 + 1 + 1 = 3 \]
\[ 3 + 3 + 1 = 7 ? \]

DAG Example

- Cover with 3 input gates
Not Optimal for DAGs

• Delay:
  – in fanout model, depends on problem you haven’t already solved (delay of node depends on number of uses)

What do people do?

• Cut DAGs at fanout nodes
• optimally solve resulting trees

• Area
  – guarantees covered once
  • get accurate costs in covering trees, made “premature” assignment of nodes to trees

• Delay
  – know where fanout is

Bounding

• Tree solution give bounds (esp. for delay)
  – single path, optimal covering for delay
  – (also make tree by replicating nodes at fanout points)
• no fanout cost give bounds
  – know you can’t do better
• delay bounds useful, too
  – know what you’re giving up for area
  – when delay matters

(Multiple Objectives?)

• Like to say, get delay, then area
  – won’t get minimum area for that delay
  – algorithm only keep best delay
  – …but best delay on off critical path piece not matter
  • …could have accepted more delay there
  – don’t know if on critical path while building subtree
  – (iterate, keep multiple solutions)

Many more details...

• Implement well

• Combine criteria

Admin

• Reading for today: blackboard
• Reading for Monday: online/Xplorer
• Office Hour: T4:30pm
  – Or make an appointment
• Project: C is common language
  – What will support
Big Ideas

- simple cost models
- problem formulation
- identifying structure in the problem
- special structure
- characteristics that make problems hard
- bounding solutions